

Carnegie Mellon  
Software Engineering Institute

# Attack Modeling for Information Security and Survivability

Andrew P. Moore  
Robert J. Ellison  
Richard C. Linger

*March 2001*

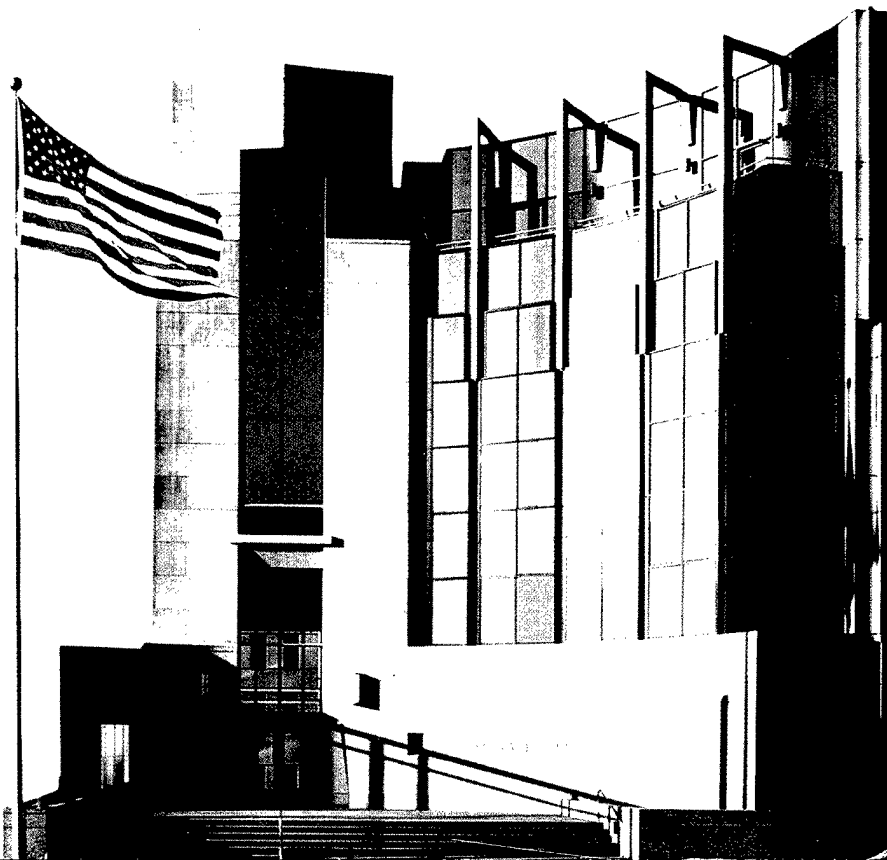
**Survivable Systems**

20010420 017

Unlimited distribution subject to the copyright

**Technical Note**  
CMU/SEI-2001-TN-001

**DISTRIBUTION STATEMENT A**  
Approved for Public Release  
Distribution Unlimited



Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

# **Attack Modeling for Information Security and Survivability**

Andrew P. Moore  
Robert J. Ellison  
Richard C. Linger

*March 2001*

**Survivable Systems**

Unlimited distribution subject to the copyright

**Technical Note**  
CMU/SEI-2001-TN-001

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2001 by Carnegie Mellon University.

#### NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (<http://www.sei.cmu.edu/publications/pubweb.html>).

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The Problem	1
1.2	ACME Enterprise	2
<b>2</b>	<b>Attack Trees</b>	<b>4</b>
2.1	Structure and Semantics	4
2.2	ACME Attack Tree	5
<b>3</b>	<b>Attack Pattern Reuse</b>	<b>8</b>
3.1	Attack Patterns	8
3.2	Attack Profiles	11
<b>4</b>	<b>Attack Tree Refinement</b>	<b>13</b>
4.1	Profile/Enterprise Consistency	14
4.2	Pattern Application	15
<b>5</b>	<b>Conclusions</b>	<b>20</b>



---

## List of Figures

Figure 1: ACME, Inc. Enterprise Architecture	3
Figure 2: High-Level Attack Tree for ACME	6
Figure 3: Web Server Attack Refinement	7
Figure 4: Buffer Overflow Attack	10
Figure 5: Unexpected Operator Attack	10
Figure 6: Internet-Based Enclave Attack Reference Model	12
Figure 7: Attack Tree Refinement Process	13
Figure 8: ACME Enterprise Intranet	14
Figure 9: PTN-Based Enclave Attack Reference Model	15
Figure 10: Buffer Overflow Attack Refinement	16
Figure 11: Applying Attack Patterns	17
Figure 12: Unexpected Operator Attack Refinement	19





---

## **Abstract**

Many engineering disciplines rely on engineering failure data to improve their designs. Unfortunately, this is not the case with information system engineers, who generally do not use security failure data—particularly attack data—to improve the security and survivability of systems that they develop. Part of the reason for this is that, historically, businesses and governments have been reticent to disclose information about attacks on their systems for fear of losing public confidence or for fear that other attackers would exploit the same or similar vulnerabilities. Specific, detailed attack data has just not been available.

However, increased public interest and media coverage of the Internet's security have resulted in increased publication of attack data in books, Internet newsgroups, and CERT security advisories, for example. Engineers can now use this data in a structured way to improve information system security and survivability.

This technical note describes and illustrates an approach for documenting attack information in a structured and reusable form. We expect that security analysts can use this approach to document and identify commonly occurring attack patterns, and that information system designers and analysts can use these patterns to develop more survivable information systems.



---

## 1 Introduction

Engineers have long relied on the analysis of engineering failures to improve their designs. Imagine what would result if bridge builders had ignored the lessons learned from the torsional oscillations that caused the Tacoma Narrows Bridge to collapse. Or if shipbuilders had ignored the lessons learned about inadequate lifeboat space that allowed the great loss of life when the Titanic sank. Engineering success requires that we also learn from less famous disasters. The aerospace community, for example, has institutionalized a means for learning from air traffic accidents that has resulted in very low risk of death during air travel, despite its inherent hazards.

### 1.1 The Problem

Information system engineers generally do not use security failure data—particularly attack data—to improve their designs and implementations. This is partly because of a lack of publicly available data [Anderson 93]. Businesses and governments are reticent to draw attention to attacks on their systems for fear that other attackers will exploit the same or similar vulnerabilities. Even after their systems are strengthened to block attacks, organizations resist divulging the attack for fear of losing public confidence.

Despite organizational reticence to disclose attacks on their systems, attack data have become more available over the past decade, primarily because of increased public interest and media coverage of Internet security. Organizations such as the Software Engineering Institute's CERT<sup>®</sup> Coordination Center were formed primarily to help protect business and government information systems from Internet-based security attacks, in part by publishing security advisories that did not disclose the names of the organizations involved. Many books on the subject of how hackers break into systems have been published [Klander 97, Scambray 01]. Nevertheless, recent research shows that information system engineers are not learning from these documented security attacks [Arbaugh 00]. Information systems being built and managed today are prone to the same or similar vulnerabilities that have plagued them for years.

Information system engineers need a better way to use and analyze attack data to learn from previous experience. This paper proposes a means to document information-security attacks in a structured and reusable form. We expect that security analysts will be able to use the structures described to identify commonly occurring attack patterns derived from real attack data.<sup>1</sup> Further, we expect that information system designers and analysts will be able to use

---

<sup>®</sup> CERT and CERT Coordination Center are registered in the U.S. Patent and Trademark Office.

<sup>1</sup> These structures are not intended for use by victims of an attack, but by analysts who more fully understand attacker profiles and the impact of specific attacks on system operation.

the attack patterns to develop more survivable systems. Future work will refine and validate the approach through its application to real-world examples.

We base our documentation approach on a structure called the *attack tree* [Schneier 99]. Section 2 describes the attack tree format and semantics. Section 3 describes a structure for capturing and reusing generic patterns of information-security attacks. Section 4 defines a model for refining attack trees that is based on the specification and reuse of these generic attack patterns. Section 5 summarizes this paper and characterizes future work.

## **1.2 ACME Enterprise**

Throughout the paper we use attacks on a fictitious company, called ACME, Inc., to illustrate concepts and issues. Figure 1 depicts the ACME enterprise environment and architecture. The important features to notice are that ACME's property is physically protected by a fence around the perimeter. The only entrance to the property is through the fenced perimeter. In addition to the perimeter fence, physical security consists of a guarded front gate. The local networks are split between the Headquarters' LAN and the Network Services' LAN. Internet users connect to the ACME Web server through a firewall. Dial-up users get access to a particular server on the Network Services' LAN.

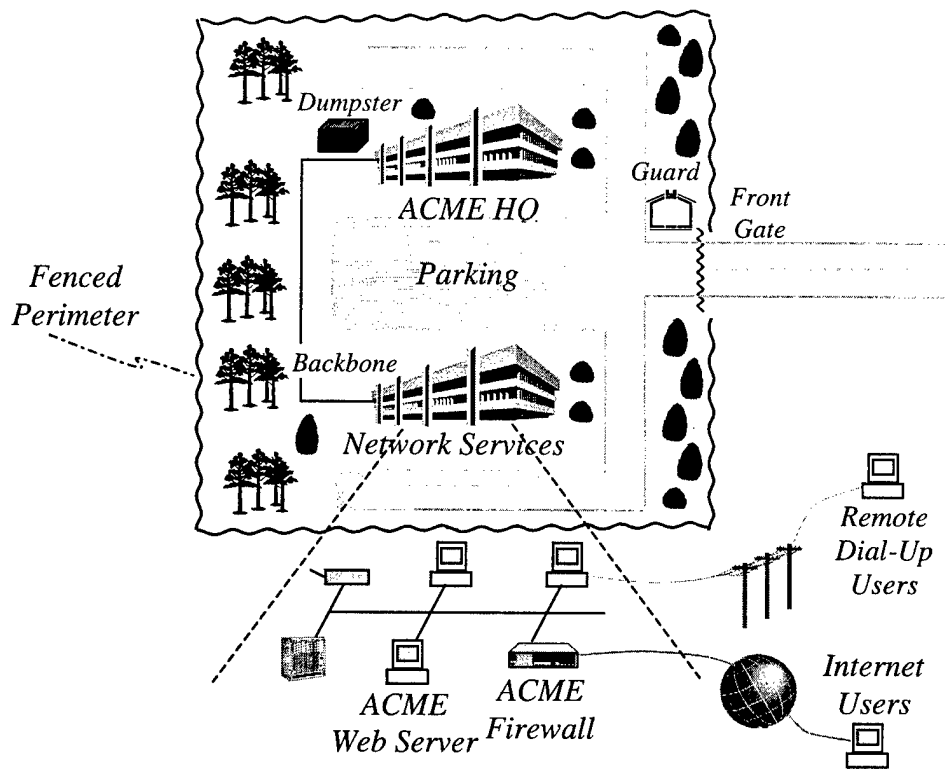


Figure 1: ACME, Inc. Enterprise Architecture

---

## 2 Attack Trees

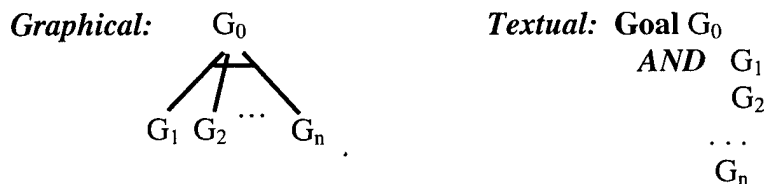
Attack trees have existed in various forms, and under various names, for many years, but have been most recently described as a systematic method to characterize system security based on varying attacks [Schneier 00]. They refine information about attacks by identifying the compromise of enterprise security or survivability as the root of the tree. The ways that an attacker can cause this compromise iteratively and incrementally are represented as lower level nodes of the tree. An enterprise typically has a set, or forest, of attack trees that are relevant to its operation. The root of each tree in a forest represents an event that could significantly harm the enterprise's mission. Each attack tree enumerates and elaborates the ways that an attacker could cause the event to occur. Each path through an attack tree represents a unique attack on the enterprise.

### 2.1 Structure and Semantics

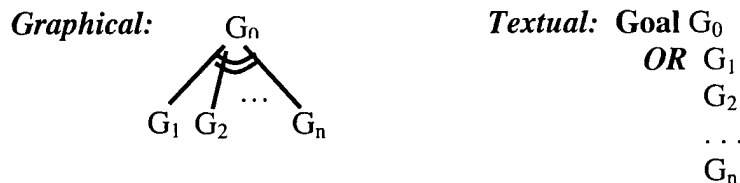
We decompose a node of an attack tree either as

- a set of attack sub-goals, all of which must be achieved for the attack to succeed, that are represented as an AND-decomposition, or
- a set of attack sub-goals, any one of which must be achieved for the attack to succeed, that are represented as an OR-decomposition.

Attack trees can be represented graphically or textually. We represent an AND-decomposition as follows:

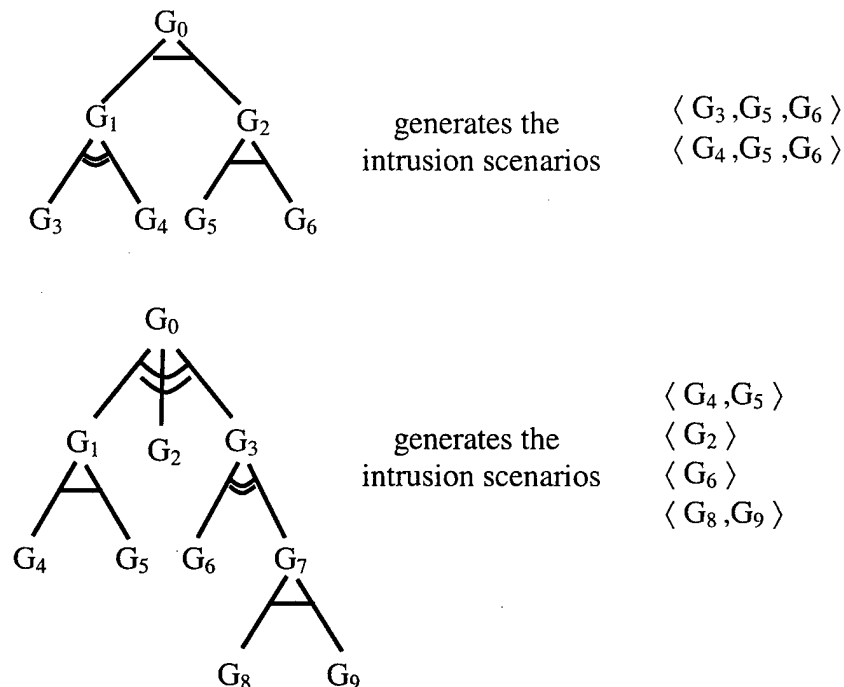


This represents a goal  $G_0$  that can be achieved if the attacker achieves each of  $G_1$  through  $G_n$ . We represent an OR-decomposition similarly:



This represents a goal  $G_0$  that can be achieved if the attacker achieves any one of  $G_1$  through  $G_n$ . Generally we use the textual representation in this paper, since the graphical representation tends to be awkward for non-trivial attack trees.

Attack trees consist of any combination of AND- and OR-decompositions. We generate individual intrusion scenarios from an attack tree by traversing the tree in a depth-first manner. For example,



In general, leaf goals are added onto the end of scenarios as they are generated. OR-decompositions cause new scenarios to be generated. AND-decompositions cause existing scenarios to be extended. Intermediate nodes of the attack tree do not appear in the intrusion scenarios because they are elaborated by lower level goals.

Attack trees allow the refinement of attacks to a level of detail chosen by the developer. They exhibit the property of referential transparency as characterized in Prowell :

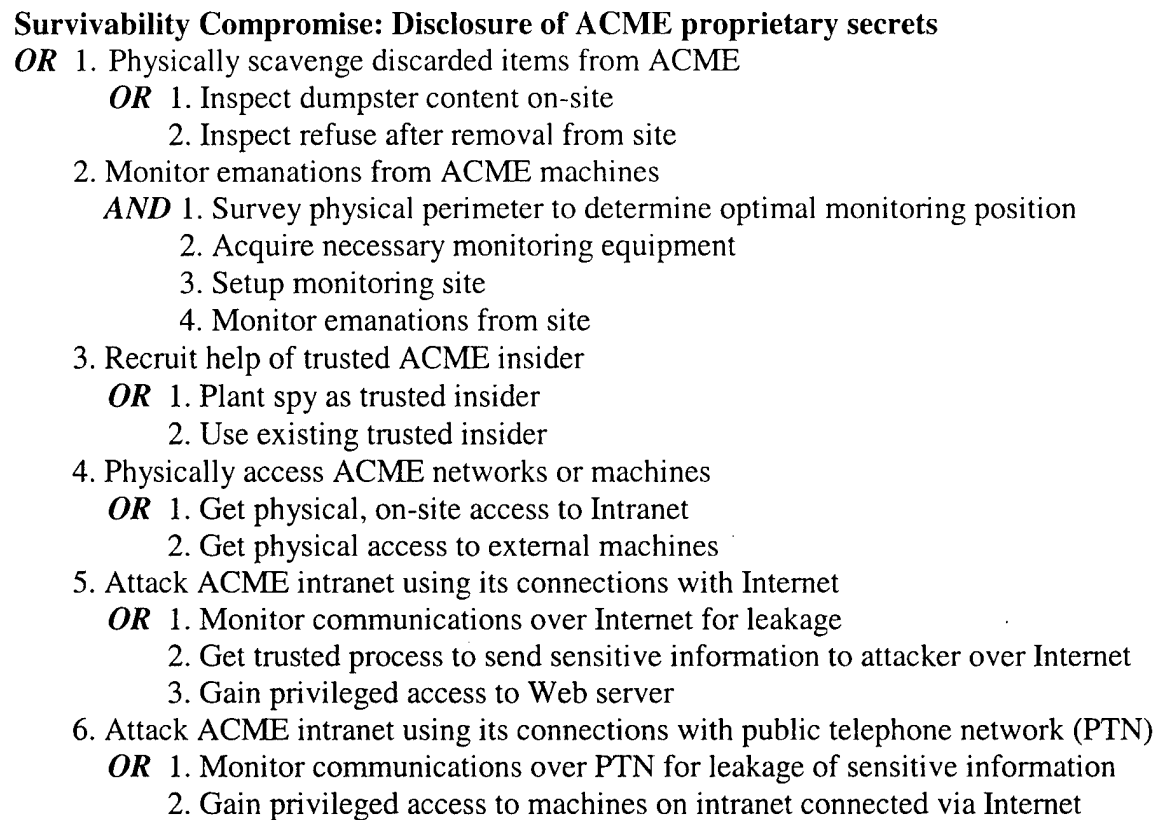
“Referential transparency implies that the relevant lower level details of an entity are abstracted rather than omitted in a particular system of higher level description, so that the higher level description contains everything needed to understand the entity when placed in a larger context” [Prowell 99].

This property permits the developer to explore certain attack paths in more depth than others, while still allowing the developer to generate intrusion scenarios that make sense. In addition, refining the branches of the attack tree generates new branches, resulting in intrusion scenarios at the new lower level of abstraction.

## 2.2 ACME Attack Tree

Figure 2 exemplifies a high-level attack tree where the root node compromise to ACME security is the disclosure of proprietary secrets. Notice that we include physical and social engineering attacks as well as technological attacks. The first branch of the top-level OR-

decomposition deals with so-called dumpster-diving attacks, both on-site and after refuse has been removed.



*Figure 2: High-Level Attack Tree for ACME*

The second branch elaborates attacks that monitor emanations (e.g., visual or electromagnetic) from locations just outside the perimeter. Branches 3 and 4 cite attacks that exploit trusted insiders and physical access (either local or remote), respectively. Finally, branches 5 and 6 characterize technological attacks over the Internet or over the public telephone network (PTN). Considering attacks that exploit both technical and non-technical weaknesses of an enterprise's operation is critical to robust information security and survivability [Schneier 00, Anderson 93].

Although the intrusion scenarios for the attack tree in Figure 2 are very high level, we will list them below to illustrate the referential transparency property of attack trees. Recall that intermediate nodes are not included in an intrusion scenario because they are completely elaborated by lower level nodes. We use the notation  $\langle i, j, k \rangle$  to represent the intrusion scenario leaf goal  $i$ , followed by step  $j$ , and followed by step  $k$ .

- $\langle 1.1 \rangle, \langle 1.2 \rangle, \langle 2.1, 2.2, 2.3, 2.4 \rangle, \langle 3.1 \rangle, \langle 3.2 \rangle$
- $\langle 4.1 \rangle, \langle 4.2 \rangle, \langle 5.1 \rangle, \langle 5.2 \rangle, \langle 5.3 \rangle, \langle 6.1 \rangle, \langle 6.2 \rangle$



Most of these scenarios are of length 1 because they are not part of an AND-decomposition. The sole exception is the AND-decomposition under the second branch of Figure 2.

Now suppose, for example, that we need to know more about attacks over the Internet on the ACME Web server (i.e., branch 5.3 of Figure 2). Figure 3 elaborates attacks on the ACME Web server that have the goal of gaining privileged access.

The scenarios for this sub-tree that lead to access to sensitive shared intranet resources directly (i.e., 5.3.5.1) are as follows (note: we omit the 5.3. prefix from each leaf label for compactness):

- $\langle 1, 2.1, 3.1, 4.1, 5.1 \rangle, \langle 1, 2.2, 3.1, 4.1, 5.1 \rangle, \langle 1, 2.3, 3.1, 4.1, 5.1 \rangle$
- $\langle 1, 2.1, 3.2, 4.1, 5.1 \rangle, \langle 1, 2.2, 3.2, 4.1, 5.1 \rangle, \langle 1, 2.3, 3.2, 4.1, 5.1 \rangle$
- $\langle 1, 2.1, 3.1, 4.2, 5.1 \rangle, \langle 1, 2.2, 3.1, 4.2, 5.1 \rangle, \langle 1, 2.3, 3.1, 4.2, 5.1 \rangle$
- $\langle 1, 2.1, 3.2, 4.2, 5.1 \rangle, \langle 1, 2.2, 3.2, 4.2, 5.1 \rangle, \langle 1, 2.3, 3.2, 4.2, 5.1 \rangle$

The set of scenarios for gaining access to sensitive data from a protected account on the Web server (5.3.5.2) is identical to the above set with 5.2 substituted for 5.1. These 24 scenarios would replace the scenario  $\langle 5.3 \rangle$  in the original list of intrusion scenarios for the attack tree in Figure 2. This expanded list represents the intrusion scenarios for the refined attack tree.

### **5.3. Gain privileged access to ACME Web server**

**AND** 1. Identify ACME domain name

2. Identify ACME firewall IP address

**OR** 1. Interrogate domain name server

2. Scan for firewall identification

3. Trace route through firewall to Web server

3. Determine ACME firewall access control

**OR** 1. Search for specific default listening ports

2. Scan ports broadly for any listening port

4. Identify ACME Web server operating system and type

**OR** 1. Scan OS services' banners for OS identification

2. Probe TCP/IP stack for OS characteristic information

5. Exploit ACME Web server vulnerabilities

**OR** 1. Access sensitive shared intranet resources directly

2. Access sensitive data from privileged account on Web server

*Figure 3: Web Server Attack Refinement*

---

## 3 Attack Pattern Reuse

The practicality of attack trees to characterize attacks on real-world systems depends on being able to reuse previously developed patterns of attack. This section describes two structures that support such reuse: an attack pattern for characterizing an individual type of attack, and an attack profile for organizing attack patterns to make it easier to search for and apply them.

### 3.1 Attack Patterns

We define an *attack pattern* as a generic representation of a deliberate, malicious attack that commonly occurs in specific contexts. Each attack pattern contains

- the overall goal of the attack specified by the pattern
- a list of preconditions for its use
- the steps for carrying out the attack
- a list of postconditions that are true if the attack is successful

The preconditions include assumptions that we make about the attacker or the state of the enterprise that are necessary for an attack to succeed. Example preconditions include the skills, resources, access, or knowledge that the attacker must possess, and the level of risk that he or she must be willing to tolerate. The postconditions include knowledge gained by the attacker and changes to the enterprise state that result from successfully carrying out the attack steps when the preconditions hold. Over the past decade, the most common form of security vulnerability has been the incorrect handling of buffer overflows by computer programs [Cowan 00]. As shown in Figure 4, when a program is invoked an activation record is added to the system's execution stack. Each activation record contains the return address when the program terminates and any local variables and buffers. In certain programs, excessively long user input can cause the internal buffer to overflow. As shown, the buffer overflow can overwrite the local variables, return pointer, and other portions of adjacent memory. An attacker can, therefore, construct the user input to change the return pointer to return to malicious code of the attacker's choosing. This malicious code runs on return with the privilege of the original program. If the program runs with administrator privilege, which is often the case, the attacker essentially has complete control of the system. This pattern of attack is captured as follows:

**Buffer Overflow Attack Pattern:**

**Goal:** Exploit buffer overflow vulnerability to perform malicious function on target system

**Precondition:** Attacker can execute certain programs on target system

**Attack:**

- AND**
1. Identify executable program on target system susceptible to buffer overflow vulnerability
  2. Identify code that will perform malicious function when it executes with program's privilege
  3. Construct input value that will force code to be in program's address space
  4. Execute program in a way that makes it jump to address at which code resides

**Postcondition:** Target system performs malicious function

The Buffer Overflow Attack demonstrates one way for an attacker to exploit with malicious intent a program's trust in user input. This is an example of a more general class of attacks called Input Validation Attacks: if the program would have validated user input, perhaps truncating it appropriately, the program would not be vulnerable to the attack. Another example in this class is the Unexpected Operator Attack. Rather than being vulnerable to excessively long input values, programs susceptible to the unexpected operator vulnerability simply do not expect that certain operators will be included in the input. For example, the program *p* in Figure 5 expects that a file name will be passed as input so that the program can use the data contained in the file for some purpose. The program vulnerability is exploited when an attacker appends the input file name with a command composition operator (";" in this example) and a malicious command (removing all files at the current directory and below). The pattern associated with this attack is similar in form to the Buffer Overflow Attack Pattern:

**Unexpected Operator Attack Pattern:**

**Goal:** Exploit unexpected operator vulnerability to perform malicious function

**Precondition:** Attacker can execute certain programs on the target system

**Attack:**

- AND**
1. Identify executable program on the target system susceptible to unexpected operator vulnerability
  2. Identify (unexpected) operator that permits composing system calls
  3. Identify system call that would perform malicious function when executed with program's privilege
  4. Construct unexpected input by composing legal input value with system call using the unexpected operator
  5. Execute program on the target system with unexpected input

**Postcondition:** Target system performs malicious function

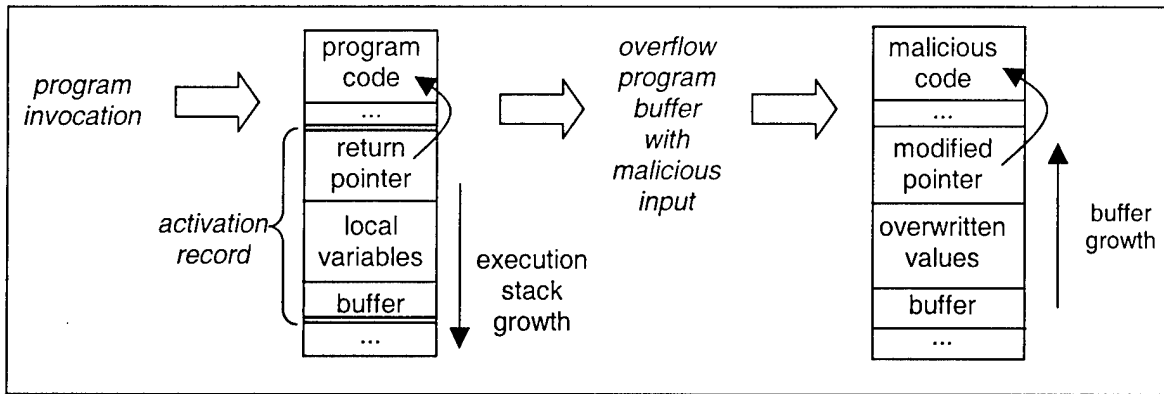


Figure 4: Buffer Overflow Attack

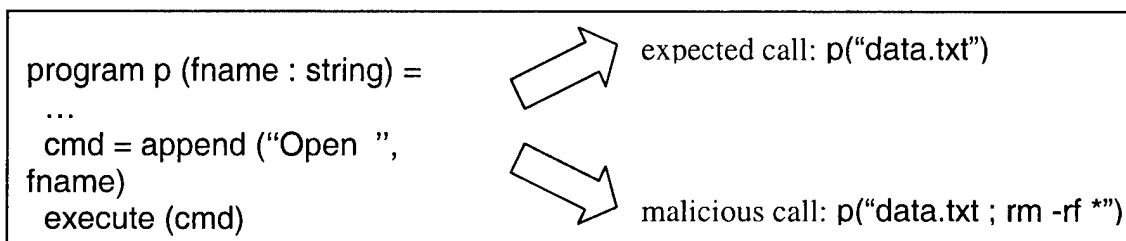


Figure 5: Unexpected Operator Attack

Attack patterns can exist at a variety of levels and do not necessarily lead to a direct compromise of information or denial of service. They may simply provide the attacker with information that he or she needs to achieve a goal. For instance, finding out the access controls that are enforced by a firewall is essential to determining how to take control of machines behind that firewall:

**Access Control Discovery Attack Pattern:**

**Goal:** Identify firewall access controls

**Precondition:** Attacker knows firewall IP address

**Attack:**

- OR** 1. Search for specific default listening ports
- 2. Scan ports broadly for any listening ports
- 3. Scan ports stealthily for listening ports
- OR** 1. Randomize target of scan
- 2. Randomize source of scan
- 3. Scan without touching target host

**Postcondition:** Attacker knows firewall access controls

Other patterns may help to satisfy the preconditions of patterns already specified. The following pattern helps to satisfy the precondition of the last pattern:

### **IP Address Discovery Attack Pattern:**

**Goal:** Identify target's firewall IP address

**Precondition:** Attacker knows target's domain name

**Attack:**

- OR*
1. Interrogate Domain Name Server
  2. Trace route through firewall to target's Web server
  3. Scan for firewall IP address

**Postcondition:** Attacker knows target's firewall IP address

## **3.2 Attack Profiles**

We further organize related attack patterns into an encompassing *attack profile*. Attack profiles contain

- a common reference model
- a set of variants
- a set of attack patterns
- a glossary of defined terms and phrases

The reference model represents an architecture template with parameters that may include specific variants. The attack patterns are also defined in terms of the variants. As we will describe more fully in the next section, attack profiles are specified independently of any particular enterprise. An enterprise whose architecture is consistent with a profile's reference model may use the profile's attack patterns, once instantiated, to help construct attack trees relevant to the enterprise's operation. Different attack profiles may address different levels of attacker access, resources, and skills, as well as different configurations of system components. Therefore, different attack profiles may help refine an enterprise-specific attack tree along different lines of attack.

Figure 6 depicts a reference model for the Internet-Based Enclave Attack Profile. The variants of this reference model are the italicized terms shown in the figure: *User*, *System*, *Intranet*, *Firewall*, and *Attacker*. That is, these elements may vary depending on the details of the specific enterprise. Attack patterns are also specified in terms of these, and potentially other, variants. The Buffer Overflow Attack Pattern is a member of this profile and can be restated in terms of the variants as follows:

**Buffer Overflow Attack Pattern:**

**Goal:** Exploit buffer overflow vulnerability to perform *malicious function* on *System*

**Precondition:** *Attacker* can execute certain programs on *System*

**Attack:**

- AND**
1. Identify executable program on *System* susceptible to buffer overflow vulnerability
  2. Identify code that will perform *malicious function* when it executes with program's privilege
  3. Construct input value that will force code to be in program's address space
  4. Execute program in way that makes it jump to address at which code resides

**Postcondition:** *System* performs *malicious function*

Notice that the profile variants *system* and *attacker* appear in the example above. Another variant, *malicious function*, does not occur as a variant of the profile's reference model. The underlined phrase is defined in the profile's glossary:

*Buffer overflow vulnerability:* A flaw in a program that, when executed with excessively long input values, causes an internal buffer to overflow overwriting portions of the execution stack and adjacent memory.

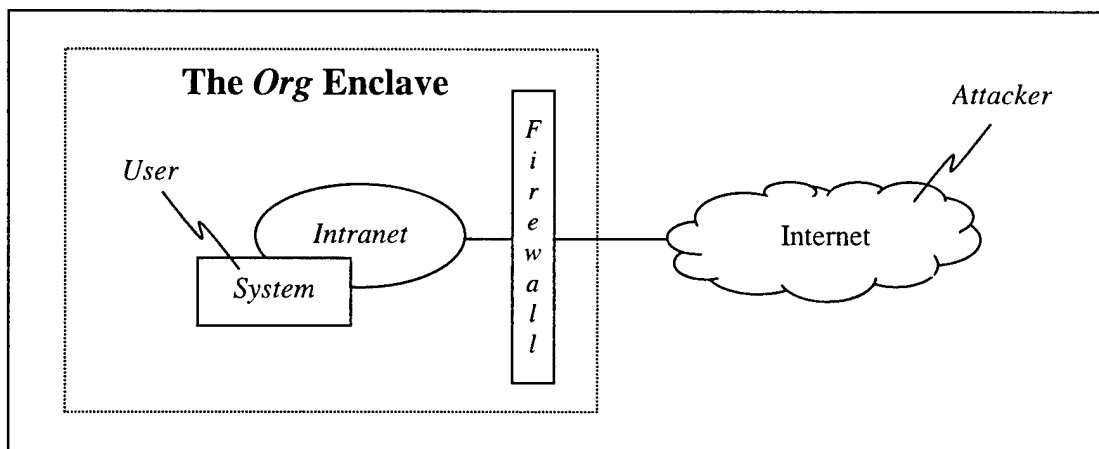


Figure 6: Internet-Based Enclave Attack Reference Model

## 4 Attack Tree Refinement

As shown in the flow chart of Figure 7, an attack tree can be refined from the root node compromise as a combination of manual extensions and pattern applications. Manual extensions depend greatly on the security expertise of the person developing the attack tree. Pattern application also depends on such expertise, but to a lesser extent. Some of this security expertise is built into an attack pattern library. Henceforth, we assume such a library already exists.

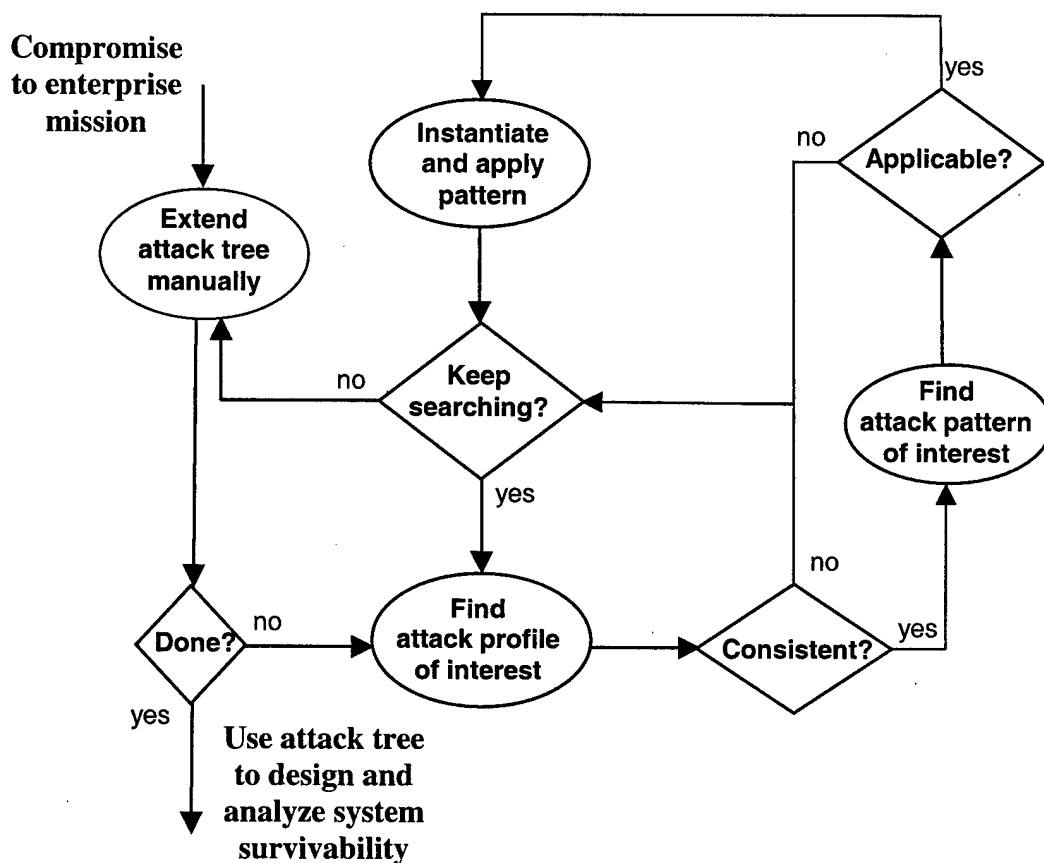


Figure 7: Attack Tree Refinement Process

A good attack pattern library provides a set of attack profiles that are rich enough to characterize the attacks that may take place on a broad range of enterprise architectures. Refining a particular enterprise's attack tree involves first finding those attack profiles that are consistent with the enterprise architecture. The developer searches the attack patterns of consistent attack profiles for a refinement of an attack path contained in the enterprise attack tree. Once found, the developer can appropriately instantiate and apply the attack pattern to

extend the enterprise attack tree. This process of pattern application intermixed with manual extension continues until the attack tree is sufficiently refined.

The rest of this section discusses in greater detail what it means for an attack profile to be consistent with an enterprise architecture, what it means for an attack pattern to be applicable to an enterprise attack tree, and how to instantiate and apply an attack pattern to refine the enterprise attack tree. The decision of when to halt the process is at the discretion of the developer.

#### 4.1 Profile/Enterprise Consistency

As we mentioned previously, the reference model associated with an attack profile can be viewed as an architecture template. The parameters of this template are the reference model variants. If a set of values exists for these variants that unifies the attack profile's reference model with some portion of the enterprise architecture, we say that the attack profile is *consistent with* the enterprise architecture. The attack patterns associated with the profile are written with respect to the profile's reference model and in terms of the profile's variants. These attack patterns are, therefore, relevant to the enterprise architecture.

As a specific example of an attack profile's consistency with an enterprise architecture, look at the Network-Based Enclave Attack Profile that we described previously. The reference model for this profile, illustrated in Figure 6, is consistent with the ACME enterprise architecture, shown Figure 1. This can be seen by instantiating the profile's variants as follows: *Org* as ACME, *Intranet* as ACME Intranet, and *Firewall* as ACME Firewall. Although the ACME Intranet was not explicitly labeled as such in Figure 1, it can be characterized as shown in Figure 8. The remaining variants associated with the reference model remain abstract, representing simply arbitrary *Users*, *Attackers*, and *Systems* to be instantiated at a later stage of refinement.

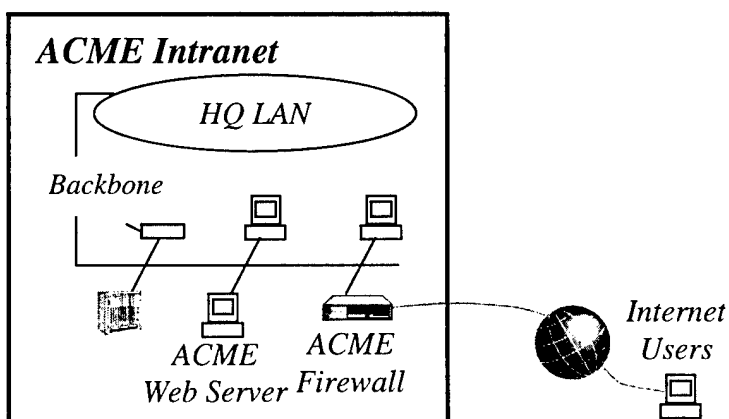


Figure 8: ACME Enterprise Intranet

Of course, other attack profiles could be used to refine the ACME attack tree, such as the PTN-Based Enclave Attack Profile. This profile contains patterns of attack over the public



telephone network through dial-up modems. The reference model for this profile, shown in Figure 9, is vulnerable to the Scan-Dialing Attack:

**Scan-Dialing Attack Pattern:**

**Goal:** Remotely login to *System*

**Precondition:** 1. *Attacker* knows *Org* telephone exchange  
2. *Attacker* knows user name of *User* on *System*

**Attack:**

**AND** 1. Scan-dial *Org* telephone exchange for answering modems  
2. Determine that connection is through *Modem* to *System*  
2. Crack *User's* password on *System*  
3. Login as *User* on *System*

**Postcondition:** *Attacker* has access to *User's* account on *System*

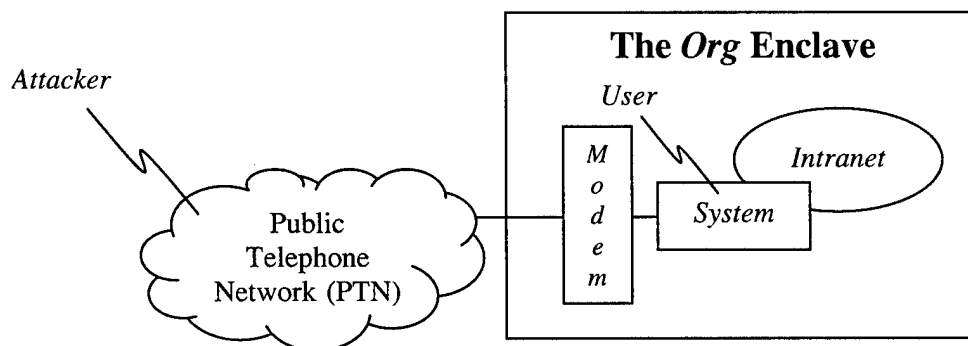


Figure 9: PTN-Based Enclave Attack Reference Model

## 4.2 Pattern Application

Determining which attack profiles are consistent with the enterprise architecture is only the first step. Analysts must also determine which attack patterns in consistent profiles help refine the enterprise attack tree. This requires identifying a pattern whose goal helps to achieve the goal identified at an attack tree node. We say that such patterns, when properly instantiated, are *applicable* to the enterprise attack tree.

For example, suppose we want to use the Buffer Overflow Attack Pattern, defined previously, to refine the ACME attack tree in Figure 3. We notice that an attacker could achieve goal 5.3.5.2 (i.e., Access sensitive data from privileged account on ACME Web server) by getting access to such a privileged account and then scanning for files that contain sensitive data. Furthermore, the attacker could achieve the first of these subgoals by exploiting a buffer overflow vulnerability on the ACME Web server. But this looks similar to the goal of the Buffer Overflow Attack Pattern.

We can use the Buffer Overflow Attack Pattern if we instantiate it so that the *System* under attack is the ACME Web server and the *malicious function* that is executed provides the attacker with access to a privileged account:

**Buffer Overflow Attack Pattern:** (instantiated)

**Goal:** Exploit buffer overflow vulnerability to get access to privileged account on ACME Web Server

**Precondition:** *Attacker* can execute certain programs on ACME Web Server

**Attack:**

- AND**
1. Identify executable program on ACME Web Server susceptible to buffer overflow vulnerability
  2. Identify code that would provide access to privileged account when executed with program's privilege
  3. Construct input value that will force code to be in program's address space
  4. Execute program in way that makes it jump to address at which code resides

**Postcondition:** *Attacker* can access privileged account

We do not require a rote substitute of the instantiations, but rather allow rewording to sound natural as long as it preserves the pattern's original intent. Figure 10 shows a refined portion of the attack tree as a result of applying the above attack pattern. Thus, we can refine the ACME attack tree in a way that permits the use of the pattern. This directed refinement of enterprise attack trees to apply a specific attack pattern is a common way to enable reuse.

**5.3.5.2 Access sensitive data from privileged account on ACME Web Server**

**AND** 1. Get access to privileged account on ACME Web server

- AND**
1. Identify executable program on ACME Web server susceptible to buffer overflow vulnerability
  2. Identify code that would provide access to privileged account when executed with program's privilege
  3. Construct input value that will force code to be in program's address space
  4. Execute program in a way that makes it jump to address at which code resides
2. Scan files for sensitive data

Figure 10: Buffer Overflow Attack Refinement

Figure 11 illustrates the three distinct types of pattern application. Each row shows the attack tree that results from applying an attack pattern, with a generic instantiation, to a particular type of node of an enterprise attack tree. Attack trees and patterns with no AND- or OR-decomposition signifier can be either an AND- or an OR-decomposition. For example, the application of an attack pattern to the leaf node of an enterprise attack tree, shown in the first row of Figure 11, does not depend on whether that node or pattern are AND- or OR-decomposed. It does depend, however, on an instantiation of the pattern that achieves the leaf node to be refined. We represent the instantiation of an attack pattern as the instantiation of each goal of the pattern, which is denoted abstractly as an "*i*" followed by the goal node instantiated. Thus, for leaf node application, the instantiation of the goal of the pattern ( $iG_R$ ) must achieve the leaf node to be refined ( $G_{K+i}$ ). The refinement of the ACME attack tree

using the Buffer Overflow Attack Pattern, shown in Figure 10, exemplifies a leaf node pattern application.

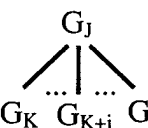
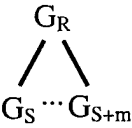
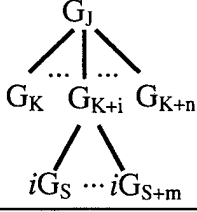
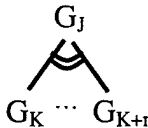
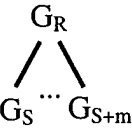
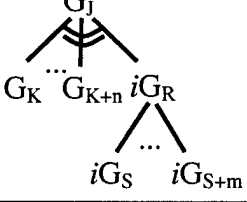

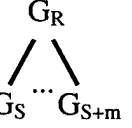
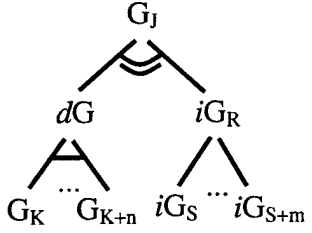
	Enterprise Attack Tree	Attack Pattern	Instantiation ( <i>i</i> ) Differentiation ( <i>d</i> )	Resulting Attack Tree
<b>Leaf Node Application</b>			$+ iG_R \text{ achieves } G_{K+i} =$	
<b>Non-Leaf Node Application to OR-Decomp</b>			$+ iG_R \text{ achieves } G_J =$	
<b>Non-Leaf Node Application to AND-Decomp</b>			$+ iG_R \text{ achieves } G_J$ $dG_J \text{ achieves } G_J =$	

Figure 11: Applying Attack Patterns

The non-leaf node applications, shown in the second and third rows of Figure 11, depend on whether the node is OR-decomposed or AND-decomposed. Applying an attack pattern at an OR-decomposed node, as in the second row, simply results in another OR-branch added to the attack tree at that node. This OR-branch represents another path for the attacker to achieve the goal  $G_J$  by means of the instantiated attack pattern. Applying a pattern at an AND-decomposed node, as in the third row, results in an attack tree with two alternative paths for the attack to achieve  $G_J$ , one by means of the original attack and one by means of the instantiated attack pattern. In this case, the user must differentiate the original attack from the new attack pattern. The differentiated goal is represented abstractly as  $dG_J$ .

To illustrate applying an attack pattern to an intermediate node of an attack tree, one would instantiate the Unexpected Operator Attack Pattern described in Section 3.1, as we did for the Buffer Overflow Attack Pattern. The resulting pattern, where the *System* under attack is the ACME Web server and the *malicious function* that is executed provides the attacker with access to a privileged account, is as follows:

**Unexpected Operator Attack Pattern:** (instantiated)

**Goal:** Exploit unexpected operator vulnerability to perform access privileged account

**Precondition:** *Attacker* can execute certain programs on ACME Web server

**Attack:**

- AND**
1. Identify executable program on ACME Web Server susceptible to unexpected operator vulnerability
  2. Identify (unexpected) operator that permits composing system calls
  3. Identify system call that would provide access to privileged account when executed with program's privilege
  4. Construct unexpected input by composing legal input value with system call using the unexpected operator
  5. Execute program on ACME Web server with unexpected input

**Postcondition:** *Attacker* can access privileged account

Figure 12 shows the attack tree of Figure 10 refined to apply this pattern at node 5.3.5.2.1.

The third row in Figure 11 represents this type of attack pattern application. Notice that we use the goal of the Buffer Overflow Attack Pattern as the differentiated goal.

**5.3.5.2 Access sensitive data from privileged account on ACME Web server**

**AND** 1. Get access to privileged account on ACME Web server

**OR** 1. Exploit buffer overflow vulnerability to access privileged account

**AND** 1. Identify executable program on ACME Web server  
susceptible to buffer overflow vulnerability

2. Identify code that would provide access to privileged  
account when executed with the program's privilege

3. Construct input value that will force code to be in the  
program's address space

4. Execute program in a way that makes it jump to address at  
which code resides

2. Exploit unexpected operator vulnerability to access privileged  
account

**AND** 1. Identify executable program on ACME Web server  
susceptible to unexpected operator vulnerability

2. Identify (unexpected) operator that permits composing  
system calls

3. Identify system call that would provide access to  
privileged account when executed with program's  
privilege

4. Construct unexpected input by composing legal input value with  
system call using the unexpected operator

5. Execute program on ACME Web server with unexpected input

2. Scan files for sensitive data

*Figure 12: Unexpected Operator Attack Refinement*

---

## 5 Conclusions

The objective of this technical note is to describe a means for documenting information-security attacks in a structured and reusable form. Within this scope, we show how to document possible attacks on an enterprise in the form of attack trees. Each attack tree enumerates and elaborates the ways that an attacker can compromise the enterprise's ability to accomplish its mission. We describe how to document and organize generic patterns of attack and how to reuse these to facilitate attack tree construction. The many examples provided illustrate the structures and techniques employed.

This paper probably raises more questions than it answers. For example:

- How does one derive requirements and improve system designs from known attacks?
- What types of analysis can one perform on attack trees?
- To what level of detail should one refine attack trees?
- At what level(s) of detail should one characterize attack patterns?
- Is there a more structured language for attack patterns that would facilitate their combination and analysis?
- How does one deal with the volatility of actual vulnerability discovery and system patching?
- How does one prioritize the branches of an attack tree according to likelihood and impact?
- How does one determine an attacker's ability and motivation for executing particular attacks?

These are a few of the questions that are driving future research. The overall goal of this research is to develop methods to derive requirements and designs for enterprise systems and operations that better survive active and malicious attacks. We believe that incorporating lessons learned from previous attacks is an important aspect of this research and that the attack tree is a useful structure to organize historical attack data. As work progresses, we will strive to

- refine the approach to facilitate more systematic analysis
- validate the practicality and scalability of the approach
- develop a broad range of attack profiles to support reuse
- formalize the model of attack tree refinement and analysis
- determine the proper role of automation to support the approach

---

## References

- Anderson 93** Anderson, R., "Why Cryptosystems Fail," in Proc. 1<sup>st</sup> Conf. On Computer and Communications Security, 1993.
- Arbaugh 00** Arbaugh, W.A., W.L. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," *IEEE Computer*, Vol. 33, No. 12, Dec. 2000.
- Cowan 00** Cowan, C., P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade," DARPA Information Survivability Conference and Expo (DISCEX), 2000.
- Klander 97** Klander, L., E.J. Renehan, Jr., "Hacker Proof: The Ultimate Guide to Network Security," Delmar Publishers, 1997.
- Prowell 99** Prowell, S.J., C.J. Trammell, R.C. Linger, and J.H. Poore, *Cleanroom Software Engineering: Technology and Process*, Addison Wesley Longman, Inc., 1999.
- Prowell 99** Prowell, S.J., C.J. Trammell, R.C. Linger, and J.H. Poore, *Cleanroom Software Engineering: Technology and Process*, Addison Wesley Longman, Inc., 1999.
- Salter 98** Salter, C., O. Saydjari, B. Schneier, J. Walner, "Toward a Secure System Engineering Methodology," Proc. Of New Security Paradigms Workshop, September 1998.
- Scambray 01** Scambray, J., S. McClure, G. Kurtz, *Hacking Exposed: Network Security Secrets & Solutions (Second Edition)*, McGraw-Hill, 2001.
- Schneier 99** Schneier, B., "Attack Trees: Modeling Security Threats," *Dr. Dobb's Journal*, December 1999.
- Schneier 00** Schneier, B., *Secrets and Lies: Digital Security in a Networked World*, John Wiley & Sons, August 2000.

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 2001	3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE Attack Modeling for Information Security and Survivability		5. FUNDING NUMBERS F19628-00-C-0003	
6. AUTHOR(S) Andrew P. Moore, Robert J. Ellison, Richard C. Linger			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2001-TN-001	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE	
13. ABSTRACT (MAXIMUM 200 WORDS) <p>Many engineering disciplines rely on engineering failure data to improve their designs. Unfortunately, this is not the case with information system engineers, who generally do not use security failure data—particularly attack data—to improve the security and survivability of systems that they develop. Part of the reason for this is that, historically, businesses and governments have been reticent to disclose information about attacks on their systems for fear of losing public confidence or for fear that other attackers would exploit the same or similar vulnerabilities. Specific, detailed attack data has just not been available.</p> <p>However, increased public interest and media coverage of the Internet's security have resulted in increased publication of attack data in books, Internet newsgroups, and CERT security advisories, for example. Engineers can now use this data in a structured way to improve information system security and survivability.</p> <p>This technical note describes and illustrates an approach for documenting attack information in a structured and reusable form. We expect that security analysts can use this approach to document and identify commonly occurring attack patterns, and that information system designers and analysts can use these patterns to develop more survivable information systems.</p>			
14. SUBJECT TERMS survivability of systems, survivability attacks, Internet security, attack patterns, information system design		15. NUMBER OF PAGES 30	
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL